

# MT365

## Audio Notes 2

---

CDA 5675 from TRACK9 pp 4 - 19

CDA 5676 pp 20 - end

This booklet contains the printed material for use with Audio-tape 3 for the audio-tape sections of *Networks 3* and Audio-tape 4 for the audio-tape sections of *Graphs 4*.

You will need to play the tape at the same time as you study the frames on the following pages.

Place the cassette player within easy reach. There are points on the tape where we have indicated that we want you to stop the tape and do some work for yourself, but you will probably find it necessary to stop the tape more often than this. Indeed, you should get into the habit of frequently stopping the tape and giving yourself time to think.

Make sure that you have paper and pencil handy before starting each tape sequence.

## BLOCK 3

### Notes for Networks 3

There are three sequences on the tape for this unit. The heading numbers below refer to the corresponding sections in *Networks 3*.

In each tape sequence we demonstrate the use of an algorithm to solve an example. In the first two tape sequences we then ask you to use the algorithm to solve a problem. There is a problem on the third algorithm in the text of the unit. Additional problems requiring the use of these three algorithms are given in the *Computer Activities Booklet*.

Each algorithm involves finding matchings in bipartite graphs, and is based on the idea of an *alternating path*, introduced in Section 2. As a reminder, this definition is given below.

#### Definition

Let  $G$  be a bipartite graph in which the set of vertices is divided into two disjoint subsets  $X$  and  $Y$ . An **alternating path** with respect to a matching  $M$  in  $G$  is a path which satisfies the conditions:

- (a) the path joins a vertex  $x$  in  $X$  to a vertex  $y$  in  $Y$ ;
- (b) the initial and final vertices  $x$  and  $y$  are not incident with an edge in  $M$ ;
- (c) alternate edges of the path are in  $M$ , and the other edges are not in  $M$ .

#### 2.2 Maximum matching algorithm

#### 3.1 Hungarian algorithm for the assignment problem

#### 4.1 Hungarian algorithm for the transportation problem

A formal statement of each algorithm is given in the unit. **You should have the unit open at the appropriate page whilst listening to each tape sequence.**

Page 14 has been left blank to enable other frames to face each other.

## BLOCK 4

### Notes for Graphs 4

There are two sequences on the tape, both associated with Section 5. The numbers 5.3 and 5.4 below refer to the corresponding sections in *Graphs 4*.

In each sequence we describe an algorithm for solving a particular problem using a branch and bound method. The problems are:

- 5.3 the knapsack problem;
- 5.4 the travelling salesman problem.

Each algorithm involves a search for an optimum solution based on the following ideas:

- a branching tree for structuring the search;
- successive improvement of a lower bound for a number to be determined.

In the case of the knapsack problem, the number to be determined is the *total value of items packed*.

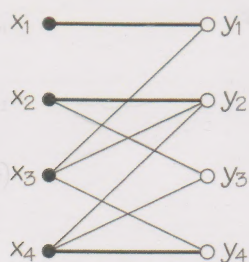
In the case of the travelling salesman problem, the number to be determined is the *total length of the route*.

## The maximum matching algorithm

### TRACK 10

#### 1 WORKED PROBLEM

Find a maximum matching in the following bipartite graph.



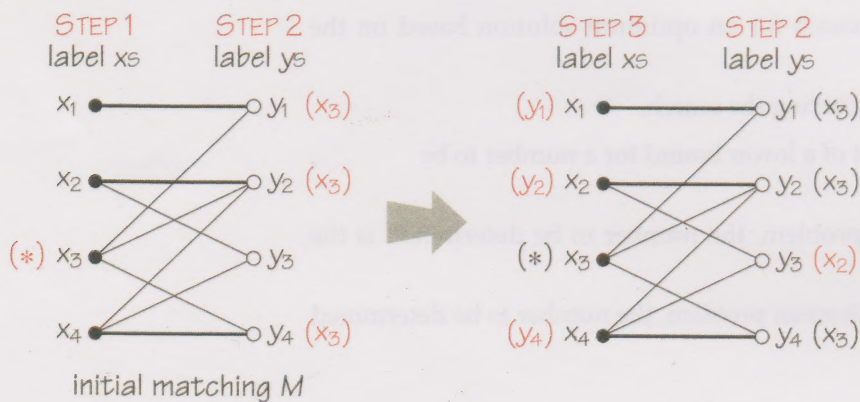
USE

- labelling procedure
- matching improvement procedure

### TRACK 11

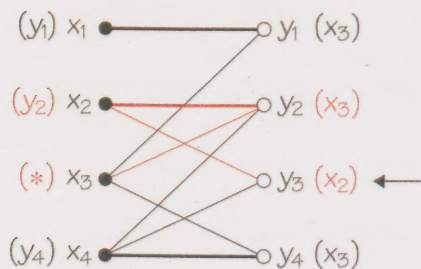
#### 2 SOLUTION TO WORKED PROBLEM

##### PART A: LABEL VERTICES



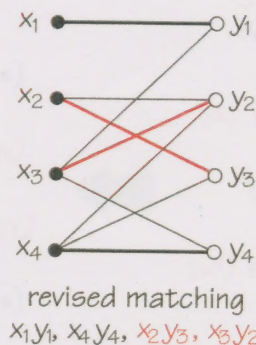
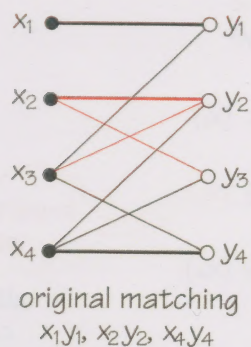
##### PART B: IMPROVE MATCHING

##### STEP 4: FIND ALTERNATING PATH



## 2 SOLUTION CONTINUED

### STEP 5: CONSTRUCT REVISED MATCHING



alternatively:  
 $x_1y_1, x_2y_2, x_3y_4, x_4y_3$

Since the revised matching has 4 edges, it is a maximum matching.

TRACK 12

## 3 SUMMARY OF THE ALGORITHM

START with any matching.

### Part A: labelling procedure

Label the vertices to identify an alternating path.

If *breakthrough* is achieved, go to Part B.

If *breakthrough* is not achieved, STOP:  
the current matching is a maximum matching.

*breakthrough occurs when a vertex in Y not incident with any edge in the current matching is labelled*

### Part B: matching improvement procedure

Find an alternating path by tracing back through the labels.

Form a new matching from:

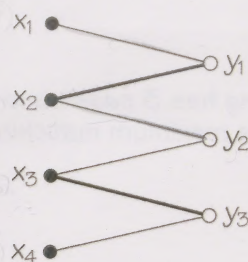
- the edges in the current matching NOT IN the alternating path,
- the edges in the alternating path NOT IN the current matching.

Return to Part A.

TRACK 13

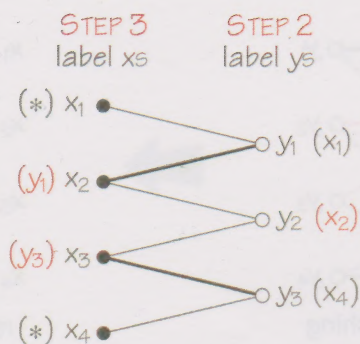
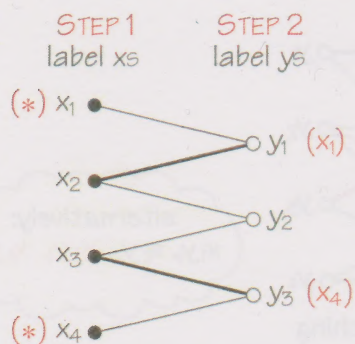
## 4 PROBLEM

Find an improved matching in the following bipartite graph.



## 5 SOLUTION

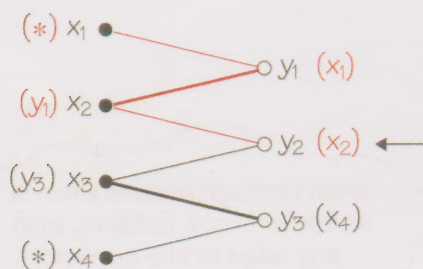
### PART A: LABEL VERTICES



breakthrough at  $y_2$   
 alternatively, could label  $y_2$  with  $x_3$

### PART B: IMPROVE MATCHING

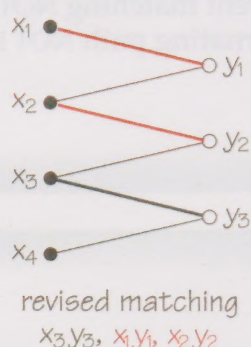
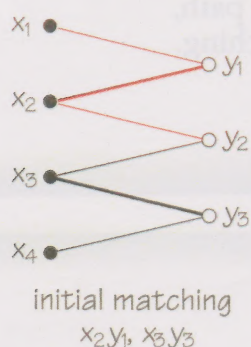
#### STEP 4: FIND ALTERNATING PATH



an alternating path is  
 $y_2x_2y_1x_1$

alternatively:  
 $y_2x_3y_3x_4$

#### STEP 5: CONSTRUCT REVISED MATCHING



alternatively:  
 $x_2y_1, x_3y_2, x_4y_3$

Since the revised matching has 3 edges and there are only 3 vertices in  $Y$ , it is a maximum matching.

## The Hungarian algorithm for the assignment problem

USE

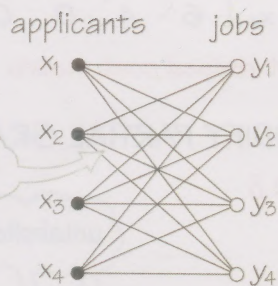
- labelling procedure
- matching improvement procedure
- modification of partial graph procedure

### 1 WORKED PROBLEM

Find the optimum assignment in the following case.

		jobs			
		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
applicants	x <sub>1</sub>	6	12	15	15
	x <sub>2</sub>	4	8	9	11
	x <sub>3</sub>	10	5	7	8
	x <sub>4</sub>	12	10	6	9

cost matrix



bipartite graph  $K_{4,4}$

## TRACK 16

### 2 SOLUTION TO WORKED PROBLEM

STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH

weights

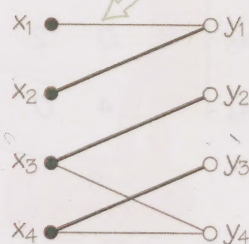
↓		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
6	x <sub>1</sub>	0	6	9	9
4	x <sub>2</sub>	0	4	5	7
5	x <sub>3</sub>	5	0	2	3
6	x <sub>4</sub>	6	4	0	3

weights →		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
↓					
6	x <sub>1</sub>	0	6	9	6
4	x <sub>2</sub>	0	4	5	4
5	x <sub>3</sub>	5	0	2	0
6	x <sub>4</sub>	6	4	0	0

first revised cost matrix

6 zeros

6 edges



maximum matching obtained by inspection

first partial graph

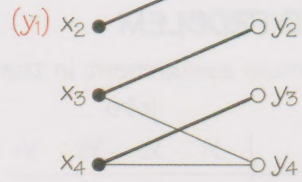
## 2 SOLUTION CONTINUED

### PART A: LABEL VERTICES

		0	0	0	3
		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
6	x <sub>1</sub>	0	6	9	6
4	x <sub>2</sub>	0	4	5	4
5	x <sub>3</sub>	5	0	2	0
6	x <sub>4</sub>	6	4	0	0

first revised cost matrix

STEPS 1, 3 STEP 2  
(\*) x<sub>1</sub> y<sub>1</sub> (x<sub>1</sub>)



labelled partial graph

no breakthrough

### PART C: MODIFY PARTIAL GRAPH

#### STEP 6: FIND $\delta$

		0	0	0	3
		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
6	x <sub>1</sub>	0	6	9	6
4	x <sub>2</sub>	0	4	5	4
5	x <sub>3</sub>	5	0	2	0
6	x <sub>4</sub>	6	4	0	0

first revised cost matrix

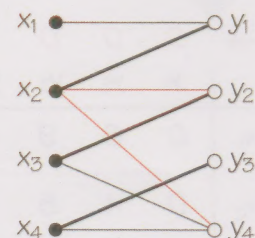
		y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
x <sub>1</sub>		6	9	6
x <sub>2</sub>		4	5	4

$\delta = 4$

#### STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH

		-4	0	0	3
		y <sub>1</sub>	y <sub>2</sub>	y <sub>3</sub>	y <sub>4</sub>
10	x <sub>1</sub>	0	2	5	2
8	x <sub>2</sub>	0	0	1	0
5	x <sub>3</sub>	9	0	2	0
6	x <sub>4</sub>	10	4	0	0

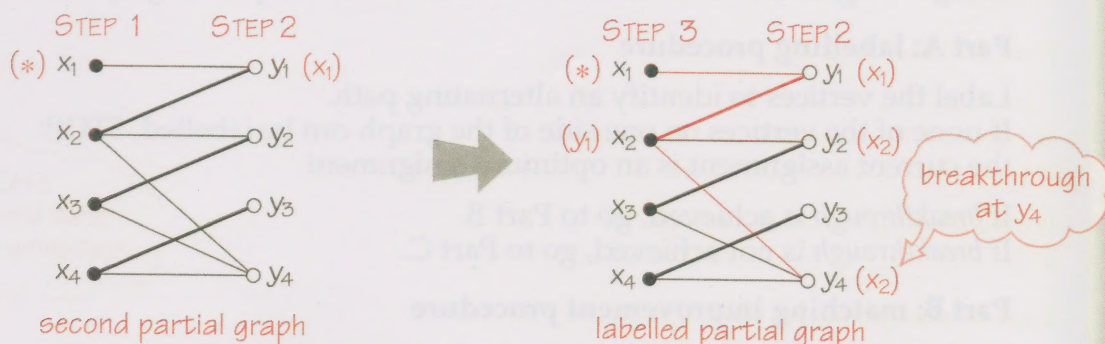
second revised cost matrix



second partial graph

## 2 SOLUTION CONTINUED

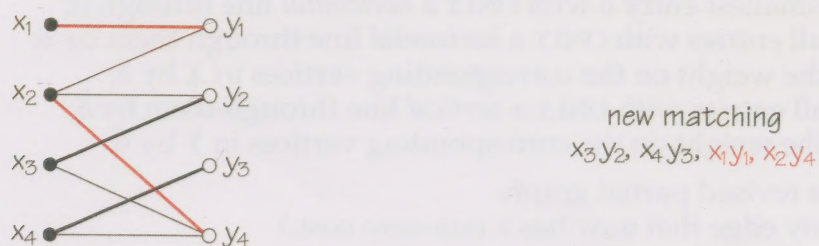
### PART A: LABEL VERTICES



### PART B: IMPROVE MATCHING

STEP 4: an alternating path is  $y_4 x_2 y_1 x_1$ .

STEP 5: revised matching is:



### PART A: LABEL VERTICES

impossible

STOP

	$y_1$	$y_2$	$y_3$	$y_4$
$x_1$	6	12	15	15
$x_2$	4	8	9	11
$x_3$	10	5	7	8
$x_4$	12	10	6	9

original cost matrix

		-4	0	0	3
		$y_1$	$y_2$	$y_3$	$y_4$
10	$x_1$	0	2	5	2
8	$x_2$	0	0	1	0
5	$x_3$	9	0	2	0
6	$x_4$	10	4	0	0

final revised cost matrix

optimum assignment:  $x_1 y_1, x_2 y_4, x_3 y_2, x_4 y_3$

total cost:  $6 + 11 + 5 + 6 = 28 = 10 + 8 + 5 + 6 - 4 + 0 + 0 + 3$

### 3 SUMMARY OF THE ALGORITHM

START with no matching.

Assign weights to the vertices and construct the first partial graph.

#### Part A: labelling procedure

Label the vertices to identify an alternating path.

If none of the vertices on one side of the graph can be labelled, STOP: the current assignment is an optimum assignment.

If *breakthrough* is achieved, go to Part B.

If *breakthrough* is not achieved, go to Part C.

SHORT CUT  
first time only – find  
matching by inspection

#### Part B: matching improvement procedure

Find an alternating path by tracing back through the labels.

Form a new matching.

Return to Part A.

#### Part C: modification of the partial graph procedure

Construct a revised cost matrix as follows.

On the existing cost matrix:

draw a *horizontal* line through each labelled vertex in  $X$ ;

draw a *vertical* line through each labelled vertex in  $Y$ ;

- find the smallest entry  $\delta$  with ONLY a *horizontal* line through it;
- decrease all entries with ONLY a *horizontal* line through them by  $\delta$ ;  
increase the weight on the corresponding vertices in  $X$  by  $\delta$ ;
- increase all entries with ONLY a *vertical* line through them by  $\delta$ ;  
decrease the weight on the corresponding vertices in  $Y$  by  $\delta$ .

Construct a revised partial graph.

(Remove any edge that now has a non-zero cost.)

Return to Part A.

### TRACK 20

### 4 PROBLEM

Find the optimum assignment  
in the following case.

	$y_1$	$y_2$	$y_3$	$y_4$
$x_1$	8	4	6	7
$x_2$	10	12	8	14
$x_3$	9	6	11	15
$x_4$	12	8	14	8

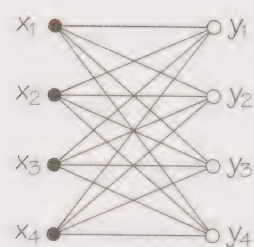
cost matrix

# 5 SOLUTION

## STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH

	$y_1$	$y_2$	$y_3$	$y_4$
$x_1$	8	4	6	7
$x_2$	10	12	8	14
$x_3$	9	6	11	15
$x_4$	12	8	14	8

cost matrix



bipartite graph  $K_{4,4}$

weights

		$y_1$	$y_2$	$y_3$	$y_4$
4	$x_1$	4	0	2	3
8	$x_2$	2	4	0	6
6	$x_3$	3	0	5	9
8	$x_4$	4	0	6	0

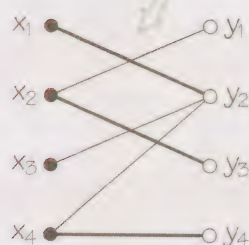
halfway cost matrix

weights →		2	0	0	0
↓		$y_1$	$y_2$	$y_3$	$y_4$
4	$x_1$	2	0	2	3
8	$x_2$	0	4	0	6
6	$x_3$	1	0	5	9
8	$x_4$	2	0	6	0

first revised cost matrix

6 zeros

6 edges



matching  
obtained  
by inspection

first partial graph

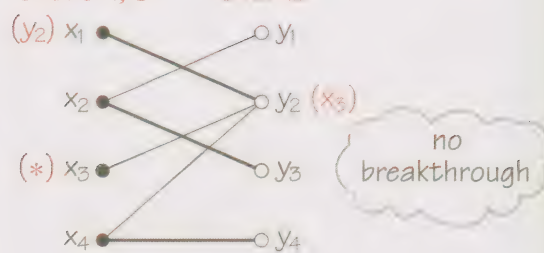
## 5 SOLUTION CONTINUED

### PART A: LABEL VERTICES

		2	0	0	0
		$y_1$	$y_2$	$y_3$	$y_4$
4	$x_1$	2	0	2	3
8	$x_2$	0	4	0	6
6	$x_3$	1	0	5	9
8	$x_4$	2	0	6	0

first revised cost matrix

STEPS 1, 3      STEP 2



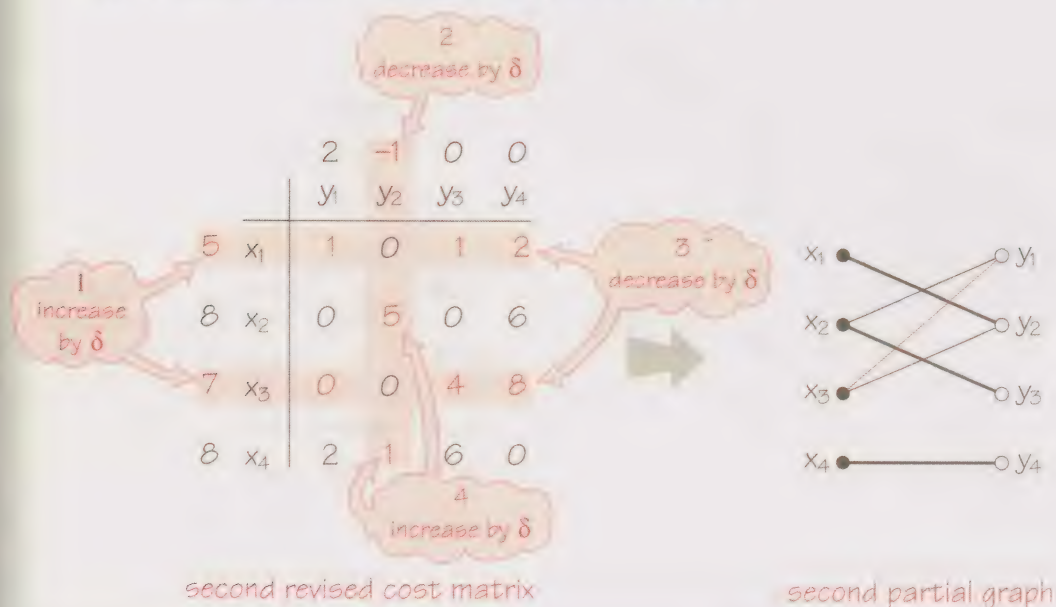
labelled partial graph

### PART C: MODIFY PARTIAL GRAPH

#### STEP 6: FIND $\delta$

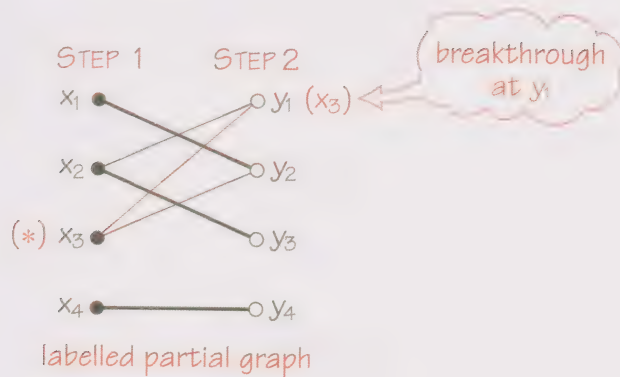


#### STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH



## 5 SOLUTION CONTINUED

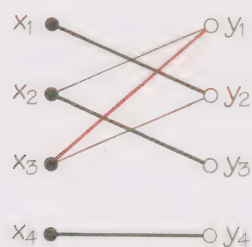
### PART A: LABEL VERTICES



### PART B: IMPROVE MATCHING

STEP 4: an alternating path is  $y_1x_3$

STEP 5: revised matching is:



new matching  
 $x_1y_2, x_2y_3, x_4y_4, x_3y_1$

### PART A: LABEL VERTICES

impossible

STOP

	$y_1$	$y_2$	$y_3$	$y_4$
$x_1$	8	4	6	7
$x_2$	10	12	8	14
$x_3$	9	6	11	15
$x_4$	12	8	14	8

original cost matrix

		2	-1	0	0
		$y_1$	$y_2$	$y_3$	$y_4$
5	$x_1$	1	0	1	2
8	$x_2$	0	5	0	6
7	$x_3$	0	0	4	8
8	$x_4$	2	1	6	0

final revised cost matrix

optimum assignment:  $x_1y_2, x_2y_3, x_3y_1, x_4y_4$

total cost:  $4 + 8 + 9 + 8 = 29 = 5 + 8 + 7 + 8 + 2 - 1 + 0 + 0$



## TRACK 23

### The Hungarian algorithm for the transportation problem

USE

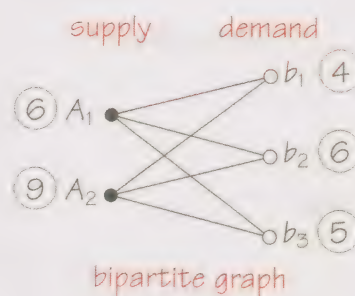
- labelling procedure
- flow-augmenting procedure
- modification of partial graph procedure

#### 1 WORKED PROBLEM

Find a minimum-cost solution to the transportation problem for the following situation.

		demand		
		4	6	5
		$b_1$	$b_2$	$b_3$
supply	6 $A_1$	3	5	7
	9 $A_2$	6	4	3

cost matrix



## TRACK 24

#### 2 SOLUTION TO WORKED PROBLEM

STEP 0: CONSTRUCT INITIAL PARTIAL GRAPH

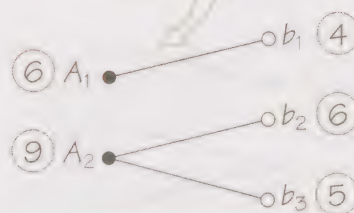
		weights			
		4	6	5	
		$b_1$	$b_2$	$b_3$	
3	6	$A_1$	0	2	4
3	9	$A_2$	3	1	0

weights → ↓			0	1	0
		4	6	5	
			$b_1$	$b_2$	$b_3$
3	6	$A_1$	0	1	4
3	9	$A_2$	3	0	0

first revised cost matrix

3 zeros

3 edges

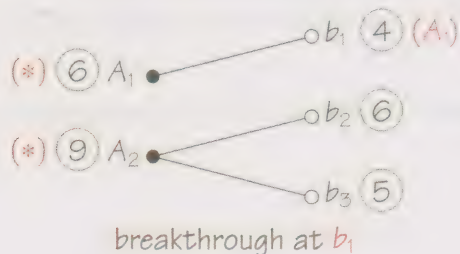


## 2 SOLUTION CONTINUED

### PART A: LABEL VERTICES

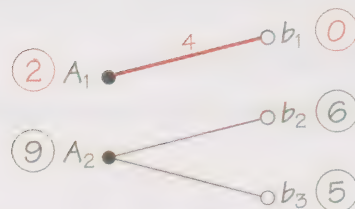
STEPS 1-3

First iteration



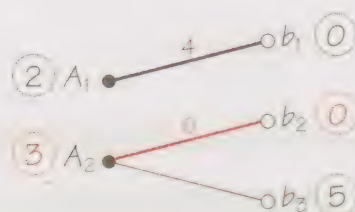
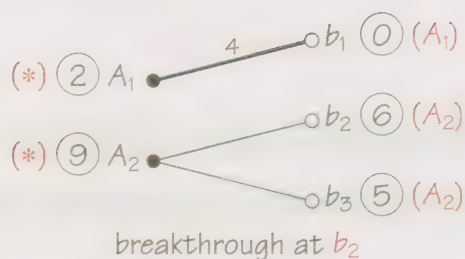
### PART B: AUGMENT FLOW

STEPS 4, 5



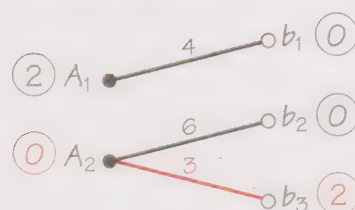
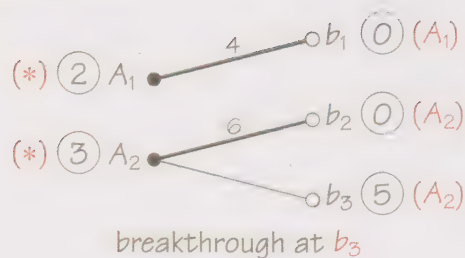
- flow-augmenting path is  $A_1b_1$
- $\min(6, 4) = 4$
- send flow of 4 down  $A_1b_1$

Second iteration



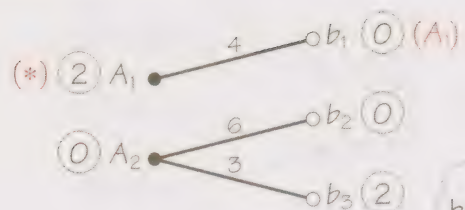
- flow-augmenting path is  $A_2b_2$
- $\min(9, 6) = 6$
- send flow of 6 down  $A_2b_2$

Third iteration



- flow-augmenting path is  $A_2b_3$
- $\min(3, 5) = 3$
- send flow of 3 down  $A_2b_3$

Fourth iteration



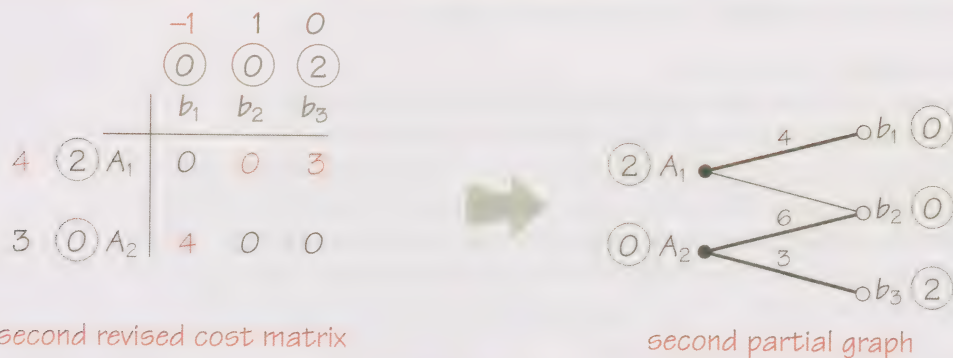
## 2 SOLUTION CONTINUED

### PART C: MODIFY PARTIAL GRAPH

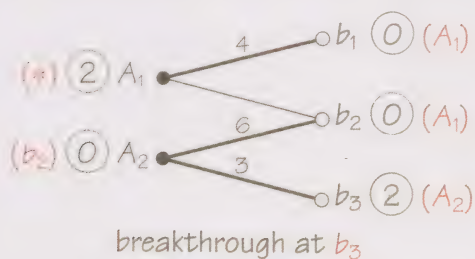
#### STEP 6: FIND $\delta$



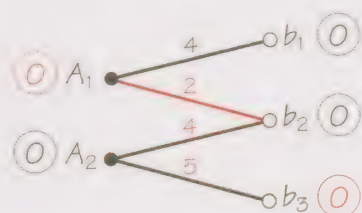
#### STEP 7: REVISE COST MATRIX AND PARTIAL GRAPH



#### PART A: LABEL VERTICES



#### PART B: AUGMENT FLOW



final flow is

4 along $A_1 b_1$	with cost	$(4 \times 3)$
2 along $A_1 b_2$		$+ (2 \times 5)$
4 along $A_2 b_2$		$+ (4 \times 4)$
5 along $A_2 b_3$		$+ (5 \times 3) = 53$

### 3 SUMMARY OF THE ALGORITHM

START with no flow.

Construct the initial partial graph.

#### Part A: labelling procedure

Label the vertices to identify a flow-augmenting path.

If no labelling is possible, STOP:  
the current solution is a minimum-cost solution.

If *breakthrough* is achieved, go to Part B.

If *breakthrough* is not achieved, go to Part C.

*breakthrough occurs when a demand vertex is labelled whose demand is not satisfied*

#### Part B: flow-augmenting procedure

Find a flow-augmenting path by tracing back through the labels.

Augment the flow.

Return to Part A.

#### Part C: modification of the partial graph procedure

Construct a new revised cost matrix as follows.

On the existing cost matrix:

draw a *horizontal* line through each labelled *supply* vertex;

draw a *vertical* line through each labelled *demand* vertex.

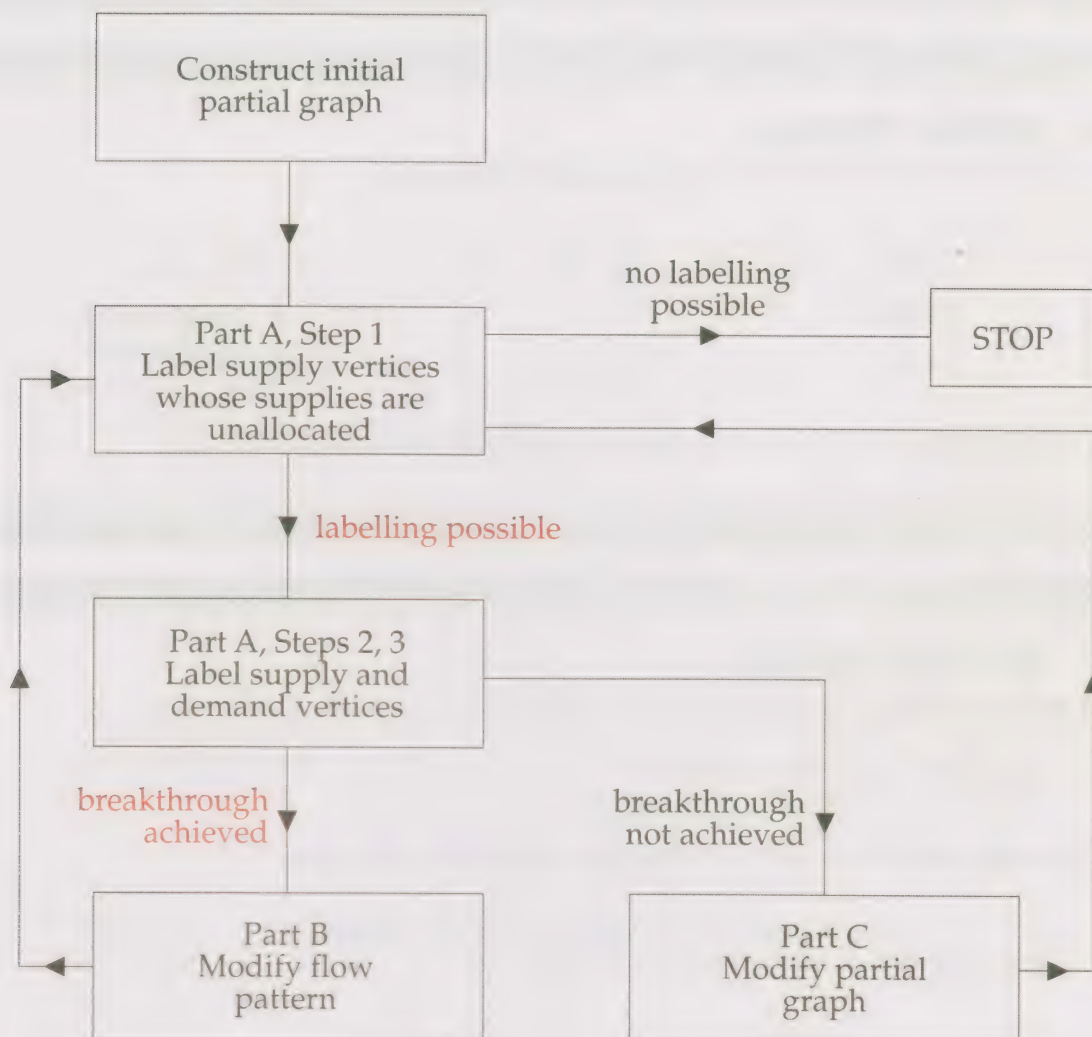
- find the smallest entry  $\delta$  with only a *horizontal* line through it;
- *decrease* all entries with only a *horizontal* line through them by  $\delta$ ;  
  *increase* the weight on the corresponding *supply* vertices by  $\delta$ ;
- *increase* all entries with only a *vertical* line through them by  $\delta$ ;  
  *decrease* the weight on the corresponding *demand* vertices by  $\delta$ .

Construct a revised partial graph.

(Remove any edge that now has a non-zero cost.)

Return to Part A.

#### 4 FLOW CHART FOR THE ALGORITHM



problem in unit

## An algorithm for the knapsack problem

### 1 WORKED PROBLEM

Consider five items with the following weights and values.

item	A	B	C	D	E
weight	4	2	7	5	1
value	3	8	9	6	1

USE

- branching tree
- lower bounds

Find a packing of greatest total value  $v$  with total weight  $w \leq 9$ .

### TRACK 02

### 2 SOLUTION VECTORS

A **solution vector** is a sequence of the form  $(x_1, x_2, x_3, x_4, x_5)$ ,

where  $\begin{cases} x_i = 1 & \text{if item } i \text{ is packed;} \\ x_i = 0 & \text{if item } i \text{ is not packed.} \end{cases}$

A **feasible solution** is one which satisfies the weight constraint.

$(1, 1, 0, 0, 1)$  corresponds to items A, B and E packed,  
with total weight  $w = 4 + 2 + 1 = 7 (\leq 9)$  ✓

feasible  
solution

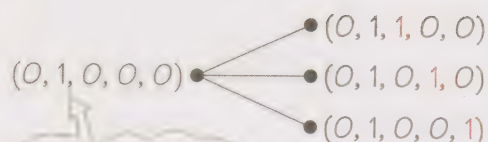
$(0, 1, 1, 0, 1)$  corresponds to items B, C and E packed,  
with total weight  $w = 2 + 7 + 1 = 10 (> 9)$  ✗

infeasible  
solution

### TRACK 03

### 3▷ BRANCHING IDEA

For example:

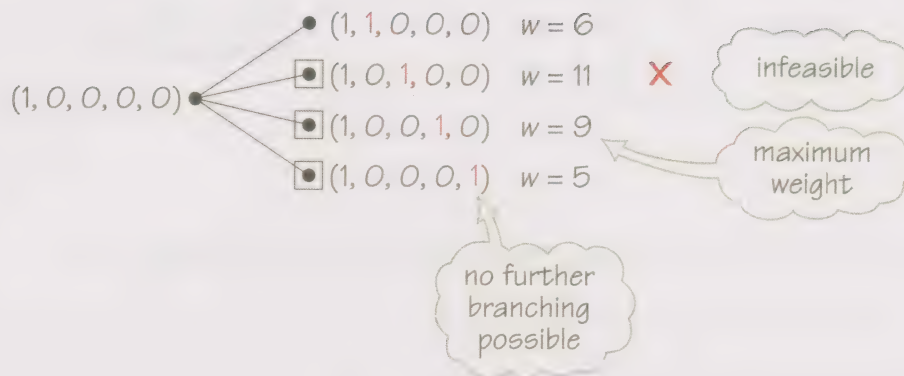


add a 1 to the  
right of this 1

these have one  
more item than  
the previous  
solution vector

## 4 DECIDING HOW TO BRANCH

For example:



Next step: branch out from  $(1, 1, 0, 0, 0)$ .

## 5 OUTLINE OF ALGORITHM

**START** with zero vector  $(0, 0, \dots, 0)$ ; feasible with value 0.  
**STORE**  $(0, 0, \dots, 0)$  and value 0.

zero vector is denoted by 0

**GENERAL STEP**

- Branch from first solution from which branching is possible.
- Calculate total weight of each new solution.
- Calculate total value of each new *feasible* solution.  
 If there is a new feasible solution with value greater than the value stored, store the new solution vector and its value instead.
- Mark vertex with  $\square$  if it corresponds to:
  - $\square$  a vector which equals or exceeds weight restriction;
  - $\square$  a vector which ends in 1.

no further branching possible from



**REPEAT** the GENERAL STEP until no more branching is possible.

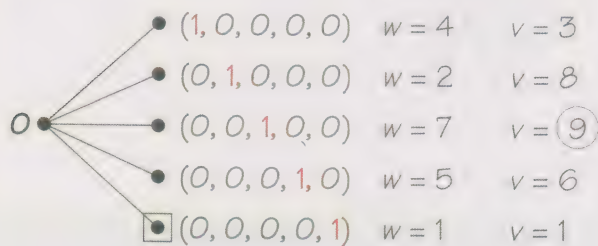
**STOP** Stored solution vector and value is optimum solution.

## 6 SOLUTION TO WORKED PROBLEM

item	A	B	C	D	E
weight	4	2	7	5	1
value	3	8	9	6	1

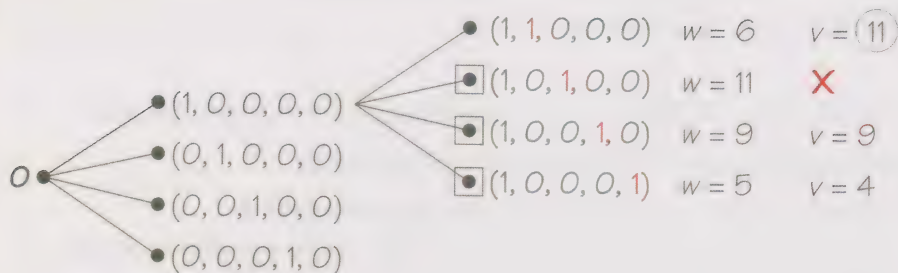
total weight  $w$   
not more than 9

First branching: from zero solution vector  $O = (0, 0, 0, 0, 0)$  with  $v = 0$ :



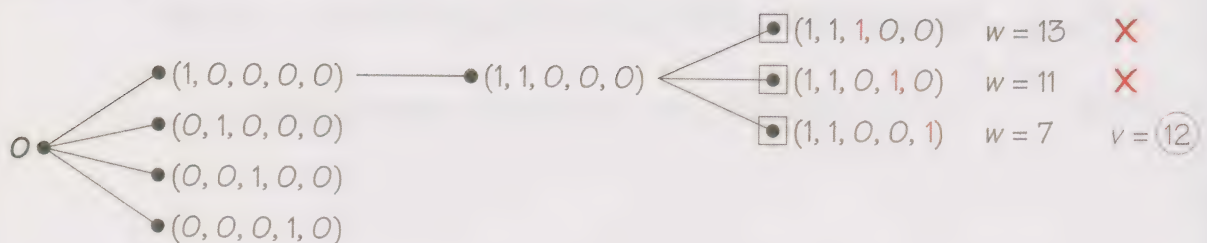
STORE  $(0, 0, 1, 0, 0)$ ,  $v = 9$

Second branching: from  $(1, 0, 0, 0, 0)$ :



STORE  $(1, 1, 0, 0, 0)$ ,  $v = 11$

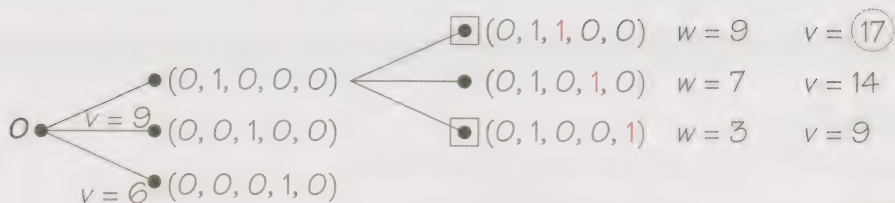
Third branching: from  $(1, 1, 0, 0, 0)$ :



STORE  $(1, 1, 0, 0, 1)$ ,  $v = 12$

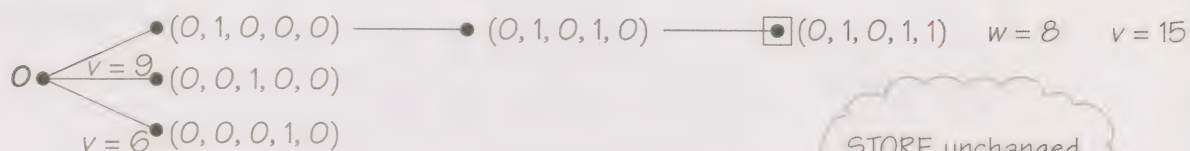
## 6 SOLUTION CONTINUED

Fourth branching: from  $(0, 1, 0, 0, 0)$ :



STORE  $(0, 1, 1, 0, 0)$ ,  $v = 17$

Fifth branching: from  $(0, 1, 0, 1, 0)$ :



STORE unchanged

Sixth branching: from  $(0, 0, 1, 0, 0)$ :



STORE unchanged

Seventh branching: from  $(0, 0, 0, 1, 0)$ :



STORE unchanged

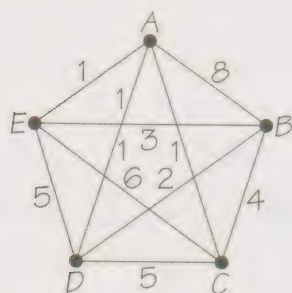
No further branching is possible: STOP.

Solution vector is  $(0, 1, 1, 0, 0)$ : items  $B$  and  $C$ , with value 17.

# An algorithm for the travelling salesman problem

## 1 WORKED PROBLEM

Find a 5-cycle through A, B, C, D, E with minimum total length:



	A	B	C	D	E
A	-	8	1	1	1
B	8	-	4	2	3
C	1	4	-	5	6
D	1	2	5	-	5
E	1	3	6	5	-

USE

- lower bounds
- branching tree

## 2 GETTING LOWER BOUND FROM TABLE

	A	B	C	D	E
A	-	8	1	1	1
B	8	-	4	2	3
C	1	4	-	5	6
D	1	2	5	-	5
E	1	3	6	5	-

need

- one entry from each row
- one entry from each column
- no 2-, 3- or 4-cycles

	A	B	C	D	E
1	A	-	7	0	0
2	B	6	-	2	0
1	C	0	3	-	4
1	D	0	1	4	-
1	E	0	2	5	4

lower bound is  $1 + 2 + 1 + 1 + 1 = 6$

		0	1	0	0	0
		A	B	C	D	E
1	A	-	6	0	0	0
2	B	6	-	2	0	1
1	C	0	2	-	4	5
1	D	0	0	4	-	4
1	E	0	1	5	4	-

new lower bound is  $6 + 1 = 7$

### 3 DECIDING HOW TO BRANCH

Consider edges with zero weight:

try to get maximum increase in lower bound

	A	B	C	D	E
A	-	6	0	0	0
B	6	-	2	0	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

exclude AC?

2

	A	B	C	D	E
A	-	6	X	0	0
B	6	-	2	0	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

lower bound increases by  $0 + 2 = 2$

	A	B	C	D	E
A	-	6	0	0	0
B	6	-	2	0	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

exclude BD?

0

	A	B	C	D	E
A	-	6	0	0	0
B	6	-	2	X	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

lower bound increases by  $1 + 0 = 1$

Label each zero with possible increase in lower bound.

Select an edge whose exclusion gives maximum increase in lower bound.

	A	B	C	D	E
A	-	6	0 <sup>2</sup>	0 <sup>0</sup>	0 <sup>1</sup>
B	6	-	2	0 <sup>1</sup>	1
C	0 <sup>2</sup>	2	-	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	4	-	4
E	0 <sup>1</sup>	1	5	4	-

maximum increase in lower bound arises from excluding AC

## 4 CARRYING OUT BRANCHING

	A	B	C	D	E
A	-	6	0 <sup>2</sup>	0 <sup>0</sup>	0 <sup>1</sup>
B	6	-	2	0 <sup>1</sup>	1
C	0 <sup>2</sup>	2	-	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	4	-	4
E	0 <sup>1</sup>	1	5	4	-

select AC

lower bound 7

include AC  
(so exclude CA)

exclude AC

cross out AC

delete row A  
and column C  
cross out CA

	A	B	D	E
B	6	-	0	1
C	X	2	4	5
D	0	0	-	4
E	0	1	4	-

	A	B	C	D	E
A	-	6	X	0	0
B	6	-	2	0	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

## 5 OUTLINE OF ALGORITHM

**START** with a given  $n \times n$  table of distances, corresponding to a complete weighted graph with  $n$  vertices.  
Carry out the initial row and column reduction, and calculate the initial lower bound.

**GENERAL STEP**

- Consider all the edges with zero weight, and choose an allowable edge  $e$  whose *exclusion* leads to the *greatest increase* in lower bound; if there are several such edges, choose the first.
- Consider the consequences of including  $e$  and excluding  $e$ . Use row and column reduction to determine these consequences, in terms of increases in the lower bound. Choose the option which gives the *smaller* lower bound; if the lower bounds are equal, *include* the edge  $e$ .  
STORE the current list of included edges, and the current lower bound.
- Continue from the current position *unless* the chosen option has a lower bound greater than a previously eliminated option, in which case backtrack to the earlier position.

CHECK  
avoid cycles  
with fewer  
than  
 $n$  edges

**REPEAT** the GENERAL STEP until a cycle with  $n$  edges has been created.

**STOP** Stored list of edges is optimum solution.

## 6 SOLUTION TO WORKED PROBLEM: FIRST BRANCHING

Consider edges with zero weight:

1	A	B	C	D	E
A	-	6	0 <sup>2</sup>	0 <sup>0</sup>	0 <sup>1</sup>
B	6	-	2	0 <sup>1</sup>	1
C	0 <sup>2</sup>	2	-	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	4	-	4
E	0 <sup>1</sup>	1	5	4	-

select AC

include AC

	A	B	D	E
B	6	-	0	1
C	X	2	4	5
D	0	0	-	4
E	0	1	4	-

exclude AC

	A	B	C	D	E
A	-	6	X	0	0
B	6	-	2	0	1
C	0	2	-	4	5
D	0	0	4	-	4
E	0	1	5	4	-

reduce row C by 2

reduce column E by 1

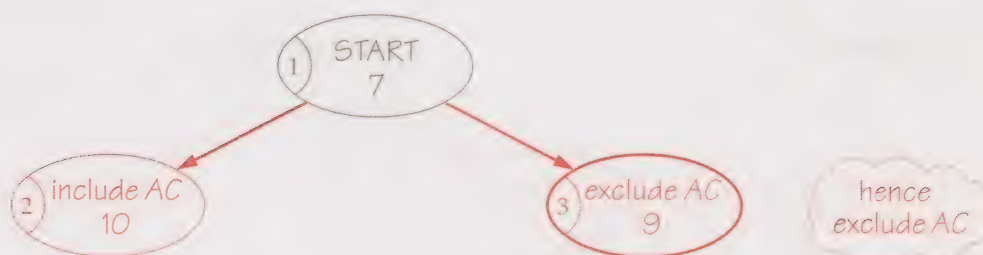
		A	B	D	E
		0	0	0	1
0	B	6	-	0	0
2	C	X	0	2	2
0	D	0	0	-	3
0	E	0	1	4	-

new lower bound is  $7 + 2 + 1 = 10$

reduce column C by 2

		A	B	C	D	E
		0	0	2	0	0
3	A	-	6	X	0	0
	B	6	-	0	0	1
	C	0	2	-	4	5
	D	0	0	2	-	4
	E	0	1	3	4	-

new lower bound is  $7 + 2 = 9$



## 7 SECOND BRANCHING

Consider edges with zero weight:

3	A	B	C	D	E
A	-	6	X	0 <sup>0</sup>	0 <sup>1</sup>
B	6	-	0 <sup>2</sup>	0 <sup>0</sup>	1
C	0 <sup>2</sup>	2	-	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	2	-	4
E	0 <sup>1</sup>	1	3	4	-

select *BC*

(lower bound increases by 2, the maximum possible)

include *BC*  
(so exclude *CB*)

4	A	B	D	E
A	-	6	0	0
C	0	X	4	5
D	0	0	-	4
E	0	1	4	-

lower bound remains 9

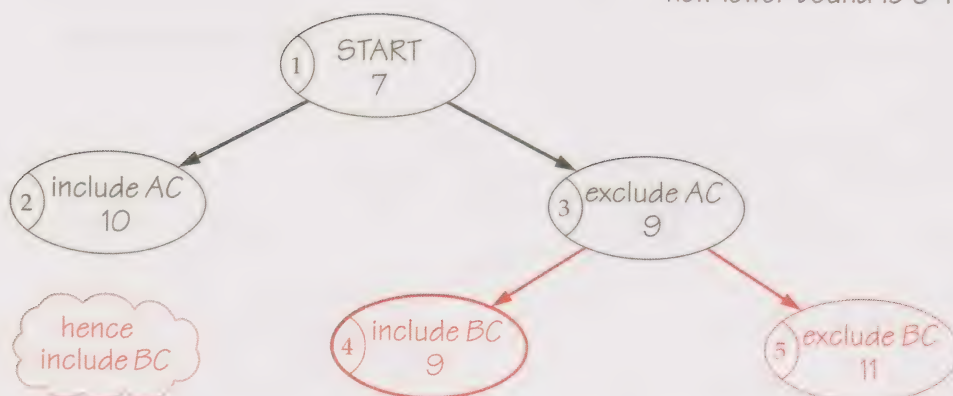
exclude *BC*

	A	B	C	D	E
A	-	6	X	0	0
B	6	-	X	0	1
C	0	2	-	4	5
D	0	0	2	-	4
E	0	1	3	4	-

reduce column *C* by 2

	0	0	2	0	0
5	A	B	C	D	E
A	-	6	X	0	0
B	6	-	X	0	1
C	0	2	-	4	5
D	0	0	0	-	4
E	0	1	1	4	-

new lower bound is  $9 + 2 = 11$



## 8 THIRD BRANCHING

Consider edges with zero weight:

4	A	B	D	E
A	-	6	0 <sup>4</sup>	0 <sup>4</sup>
C	0 <sup>4</sup>	X	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	-	4
E	0 <sup>1</sup>	1	4	-

select AD

(lower bound increases by 4, the maximum possible)

include AD  
(so exclude DA)

	A	B	E
C	0	X	5
D	X	0	4
E	0	1	-

exclude AD

	A	B	D	E
A	-	6	X	0
C	0	X	4	5
D	0	0	-	4
E	0	1	4	-

reduce column E by 4

0 0 4

6	A	B	E
C	0	X	1
D	X	0	0
E	0	1	-

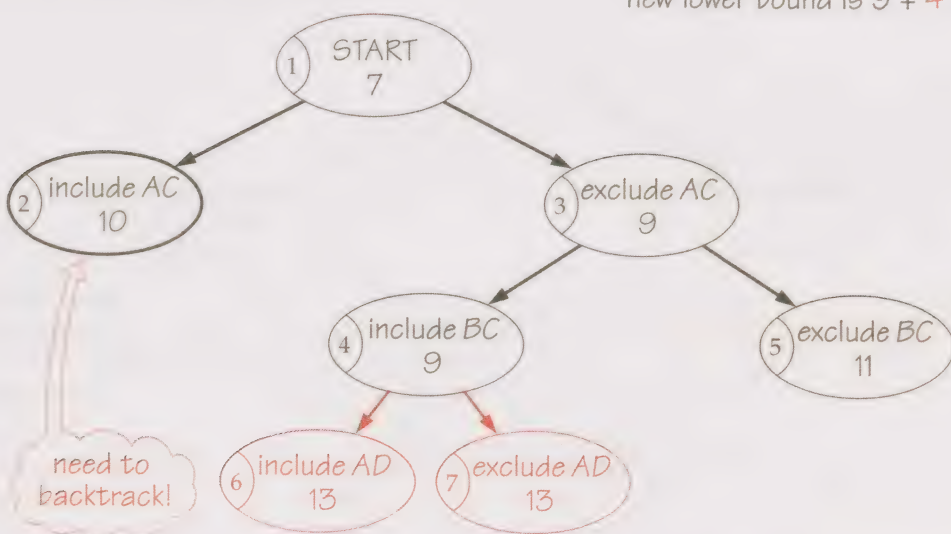
new lower bound is  $9 + 4 = 13$

reduce column D by 4

0 0 4 0

7	A	B	D	E
A	-	6	X	0
C	0	X	0	5
D	0	0	-	4
E	0	1	0	-

new lower bound is  $9 + 4 = 13$



## 9 FOURTH BRANCHING

Consider edges with zero weight:

2	A	B	D	E
B	6	-	0 <sup>2</sup>	0 <sup>2</sup>
C	X	0 <sup>2</sup>	2	2
D	0 <sup>0</sup>	0 <sup>0</sup>	-	3
E	0 <sup>1</sup>	1	4	-

select **BD**

(lower bound increases by 2, the maximum possible)

include **BD**  
(so exclude **DB**)

	A	B	E
C	X	0	2
D	0	X	3
E	0	1	-

exclude **BD**

	A	B	D	E
B	6	-	X	0
C	X	0	2	2
D	0	0	-	3
E	0	1	4	-

reduce column E by 2

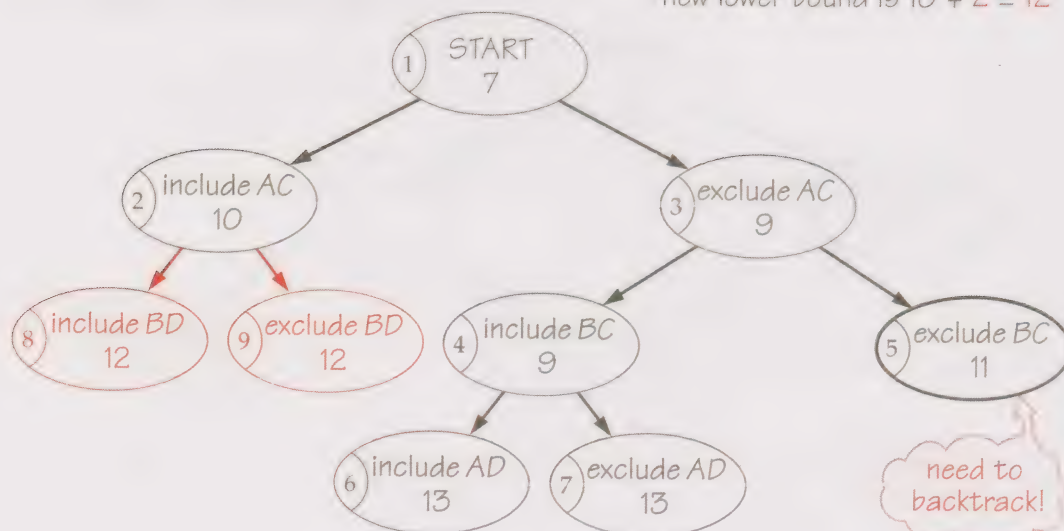
	A	B	E
C	X	0	0
D	0	X	1
E	0	1	-

new lower bound is  $10 + 2 = 12$

reduce column D by 2

	A	B	D	E
B	6	-	X	0
C	X	0	0	2
D	0	0	-	3
E	0	1	2	-

new lower bound is  $10 + 2 = 12$



10 FIFTH BRANCHING

Consider edges with zero weight:

5	A	B	C	D	E
A	-	6	X	0 <sup>0</sup>	0 <sup>1</sup>
B	6	-	X	0 <sup>1</sup>	1
C	0 <sup>2</sup>	2	-	4	5
D	0 <sup>0</sup>	0 <sup>1</sup>	0 <sup>1</sup>	-	4
E	0 <sup>1</sup>	1	1	4	-

select CA  
(lower bound increases by 2,  
the maximum possible)

include CA

	B	C	D	E
A	6	X	0	0
B	-	X	0	1
D	0	0	-	4
E	1	1	4	-

exclude CA

	A	B	C	D	E
A	-	6	X	0	0
B	6	-	X	0	1
C	X	2	-	4	5
D	0	0	0	-	4
E	0	1	1	4	-

reduce row E by 1

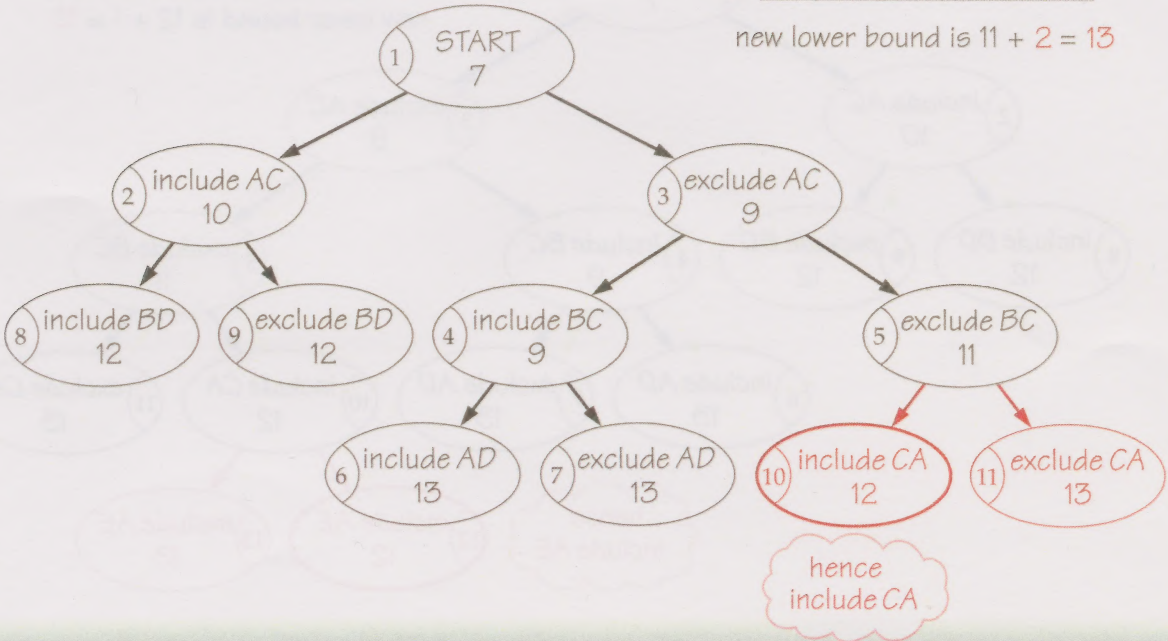
10	B	C	D	E	
0	A	6	X	0	0
0	B	-	X	0	1
0	D	0	0	-	4
1	E	0	0	3	-

new lower bound is  $11 + 1 = 12$

reduce row C by 2

	11	A	B	C	D	E
0	A	-	6	X	0	0
0	B	6	-	X	0	1
2	C	X	0	-	2	3
0	D	0	0	0	-	4
0	E	0	1	1	4	-

new lower bound is  $11 + 2 = 13$



# 11 SIXTH BRANCHING

Consider edges with zero weight:

10	B	C	D	E
A	6	X	0 <sup>0</sup>	0 <sup>1</sup>
B	-	X	0 <sup>1</sup>	1
D	0 <sup>0</sup>	0 <sup>0</sup>	-	4
E	0 <sup>0</sup>	0 <sup>0</sup>	3	-

select **AE**  
(lower bound increases by 1,  
the maximum possible)



include AE  
(so exclude EC to  
avoid 3-cycle AECA)

12	B	C	D
B	-	X	0
D	0	0	-
E	0	X	3

lower bound remains 12

exclude AE

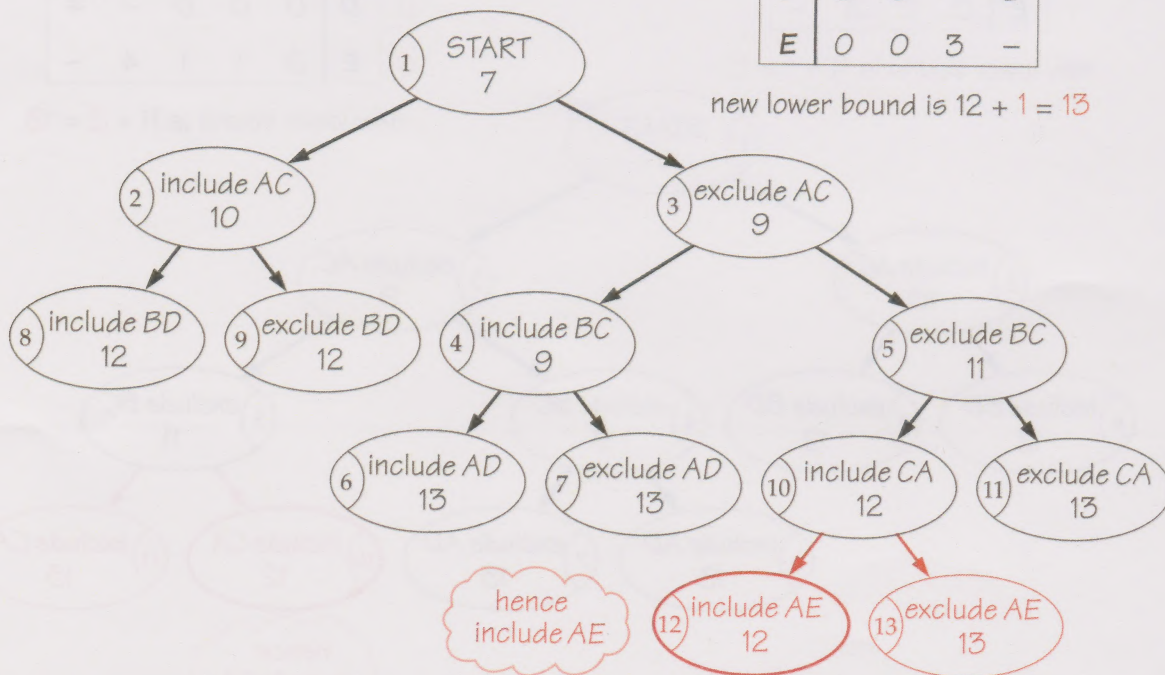
	B	C	D	E
A	6	X	0	X
B	-	X	0	1
D	0	0	-	4
E	0	0	3	-

reduce column E by 1

0 0 0 1

13	B	C	D	E
A	6	X	0	X
B	-	X	0	0
D	0	0	-	3
E	0	0	3	-

new lower bound is  $12 + 1 = 13$



## 12 FINAL BRANCHINGS

Consider edges with zero weight:

12	B	C	D
B	-	X	0 <sup>3</sup>
D	0 <sup>0</sup>	0 <sup>0</sup>	-
E	0 <sup>3</sup>	X	3

select **BD**  
(lower bound increases by 3,  
the maximum possible)

include **BD**  
(so exclude **DB**)

14	B	C
D	X	0
E	0	X

lower bound remains 12

Include **DC** and **EB**  
lower bound remains 12

exclude **BD**

15	B	C	D
B	-	X	X
D	0	0	-
E	0	X	3

not possible!  
(no route out of B)

required 5-cycle has edges  
**CA, AE, EB, BD, DC**  
and weight 12

